

Hashing and Hashtable

Hashing is a very common technique for storing data in such a way the data can be inserted and retrieved very quickly. Hashing uses a data structure called a *hash table*. Although hash tables provide fast insertion, deletion, and retrieval, operations that involve searching, such as finding the minimum or maximum value, are not performed very quickly.

AN OVERVIEW OF HASHING

A hash table data structure is designed around an array. The array consists of elements 0 through some predetermined size, though we can increase the size later if necessary. Each data item is stored in the array based on some piece of the data, called the *key*. To store an element in the hash table, the key is mapped into a number in the range of 0 to the hash table size using a function called a *hash function*.

Add the Digits

A simple approach to converting a word to a number might be to simply add the code numbers for each character. Say we want to convert the word *cats* to a number. First we convert the characters to digits using our homemade code:

c = 3
a = 1
t = 20
s = 19

Then we add them:

$$3 + 1 + 20 + 19 = 43$$

Thus in our dictionary the word *cats* would be stored in the array cell with index 43. All the other English words would likewise be assigned an array index calculated by this process.

our first attempt at converting words to numbers leaves something to be desired. Too many words have the same index. (For example, *was*, *tin*, *give*, *tend*, *moan*, *tick*, *bails*, *dredge*, and hundreds of other words add to 43, as *cats* does.) We conclude that this approach doesn't discriminate enough, so the resulting array has too few elements. We need to spread out the range of possible indices.

Choosing a Hash Function

The ideal goal of the hash function is to store each key in its own cell in the array. However, because there are an unlimited number of possible keys and a finite number of array cells, a more realistic goal of the hash function is to attempt to distribute the keys as evenly as possible among the cells of the array.

Even with a good hash function, as you have probably guessed by now, it is possible for two keys to hash to the same value. This is called a *collision* and we have to have a strategy for dealing with collisions when they occur. We'll discuss this in detail in the following.

The last thing we have to determine is how large to dimension the array used as the hash table. First, it is recommended that the array size be a prime number. We will explain why when we examine the different hash functions. After that, there are several different strategies for determining the proper array size, all of them based on the technique used to deal with collisions

Choosing a hash function depends on the data type of the key you are using. If your key is an integer, the simplest function is to return the key modulo the size of the array. There are circumstances when this method is not recommended, such as when the keys all end in zero and the array size is 10. This is one reason why the array size should always be prime. Also, if the keys are random integers then the hash function should more evenly distribute the keys.

In many applications, however, the keys are strings. Choosing a hash function to work with keys is more difficult and should be chosen carefully. A simple function that at first glance seems to work well is to add the ASCII values of the letters in the key. The hash value is that value modulo the array size. The following program demonstrates how this function works:

```

#include "stdafx.h"
#include <iostream>
#include <ctime>
#include <string>
using namespace std;
int prime=10007;
static void ShowDistrib(string[]);
static int SimpleHash(string);
void main()
{
    string names[10007];
    string name;
    string somenames[12]={ "adel", "dael", "David", "Jennifer", "Mike",
                           "Donnie", "Mayo", "Raymond", "Bernica",
                           "Clayton", "Beata", "Michael" };

    int hashval;
    for(int i = 0; i < 12; i++)
    {
        name = somenames[i];
        hashval = SimpleHash(name);
        names[hashval] = name;
    }
    ShowDistrib(names);
}
static int SimpleHash(string s)
{
    int tot = 0;
    for(int i = 0; i <= s.size(); i++)
        tot += tot + int(s[i]);
    tot= tot % prime;
    return (int)tot;
}

static void ShowDistrib(string arr[])
{
    for(int i = 0; i < prime; i++)
    {
        if(arr[i]!="")
            cout<< i << " " << arr[i] << "\n";
    }
}

```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains a list of names and IDs, each on a new line. The list is as follows:

ID	Name
1159	Donnie
1736	Bernica
2106	Clayton
2702	Mike
2714	Mayo
2790	Michael
2972	adel
2996	dael
3834	Raymond
5162	Beata
5292	David
5532	Jennifer

Below the list, the text 'Press any key to continue . . .' is displayed. The window has a standard Windows XP-style interface with a scroll bar on the right and a taskbar at the bottom.

Dr. Magdi EL-Sharkawi